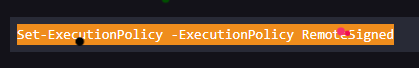
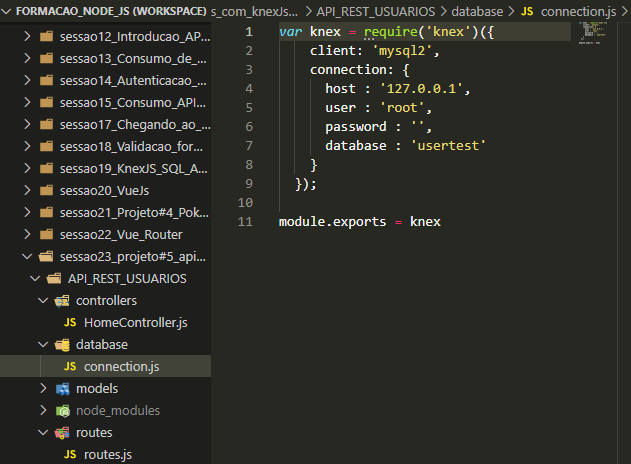
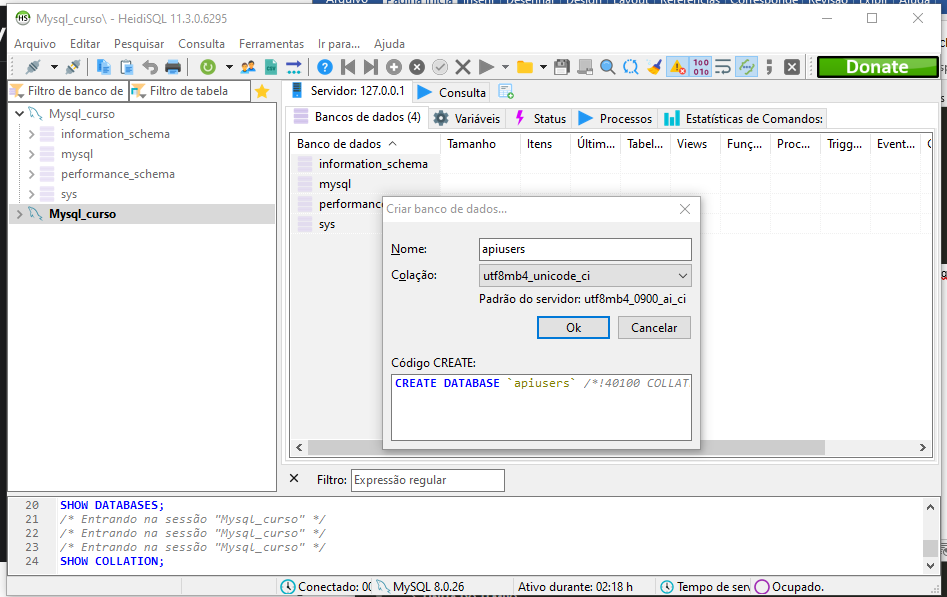
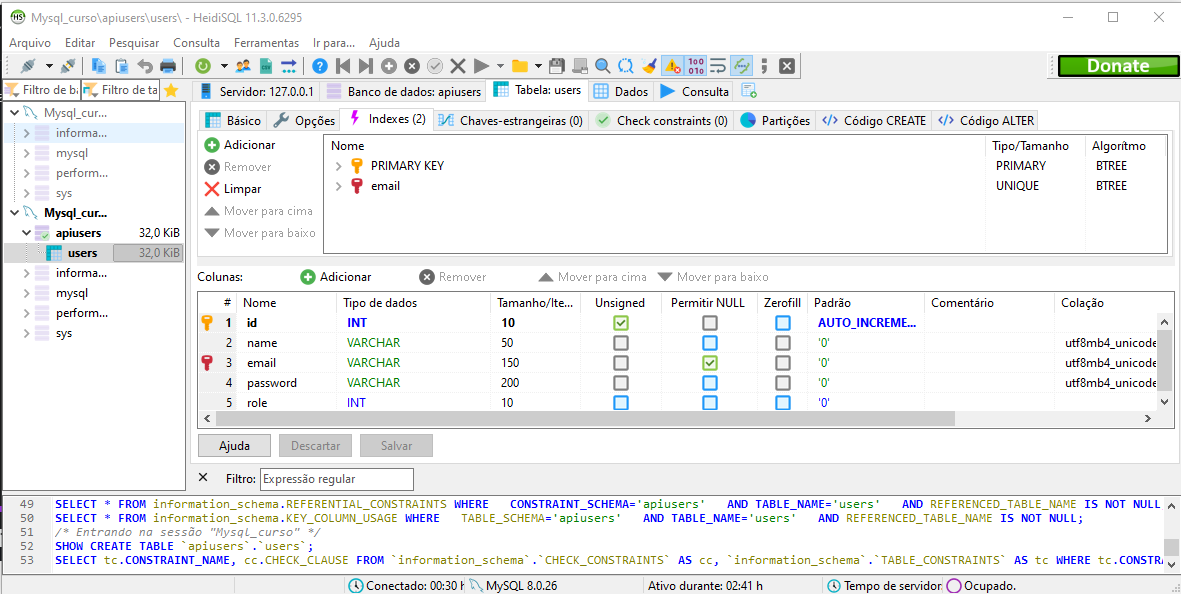
**SISTEMA DE GESTÃO DE USUÁRIOS**

1. Neste projeto iremos aprender a logica para implementação de um sistema de gestão de usuário completo, que pode ser utilizado em qualquer projeto futuro. Com cadastro, alterações, recuperação de senha etc...
2. Para agilizar a criação de um projeto foi criado a base de um projeto node no git hub para agilizar a criação
3. Link: github.com/techtuxbr/express-js-rest-api-base
4. Baixar projeto como ZIP, descompactar na pasta do projeto que será criado
5. Entrar na pasta do projeto e digitar : npm install
6. No arquivo index.js estamos importando quatro módulos
7. - boyParser que é utilizado para receber dados json via post.
8. – express
9. – router
10. Todas as rotas da aplicação serão definidas na pasta routes
11. Todos os códigos que se comunicam com o banco de dados será construído dentro da pasta models.
12. Para permitir código no powershell do Windows entre no powershell no modo administrador e altere a politica de segurança digitando:
13. 

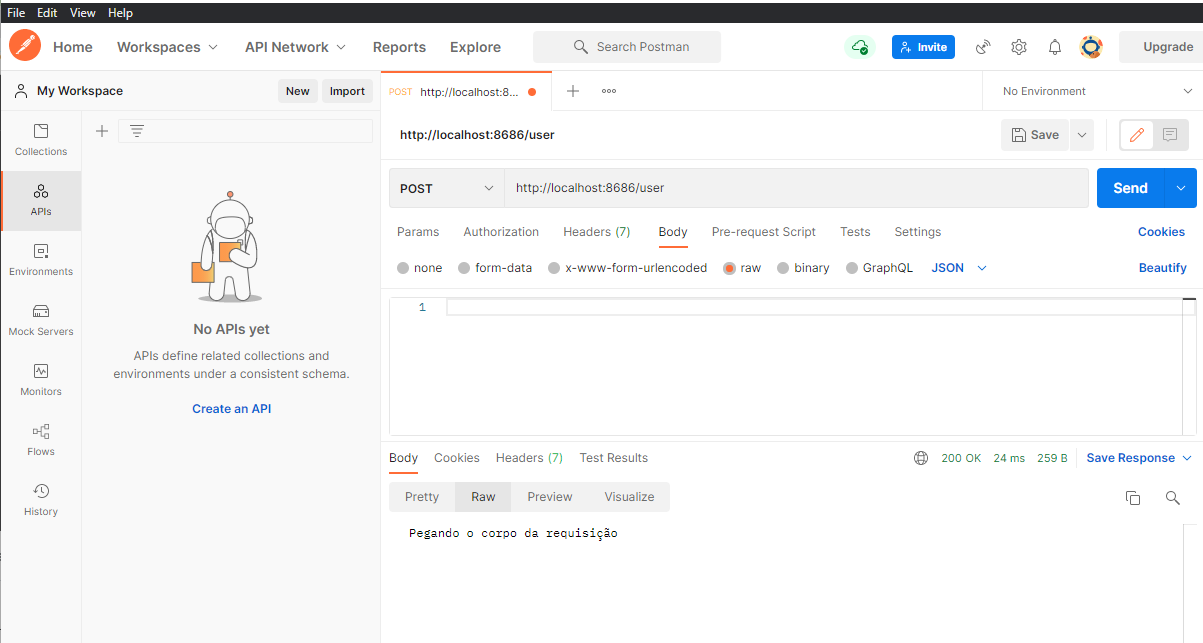
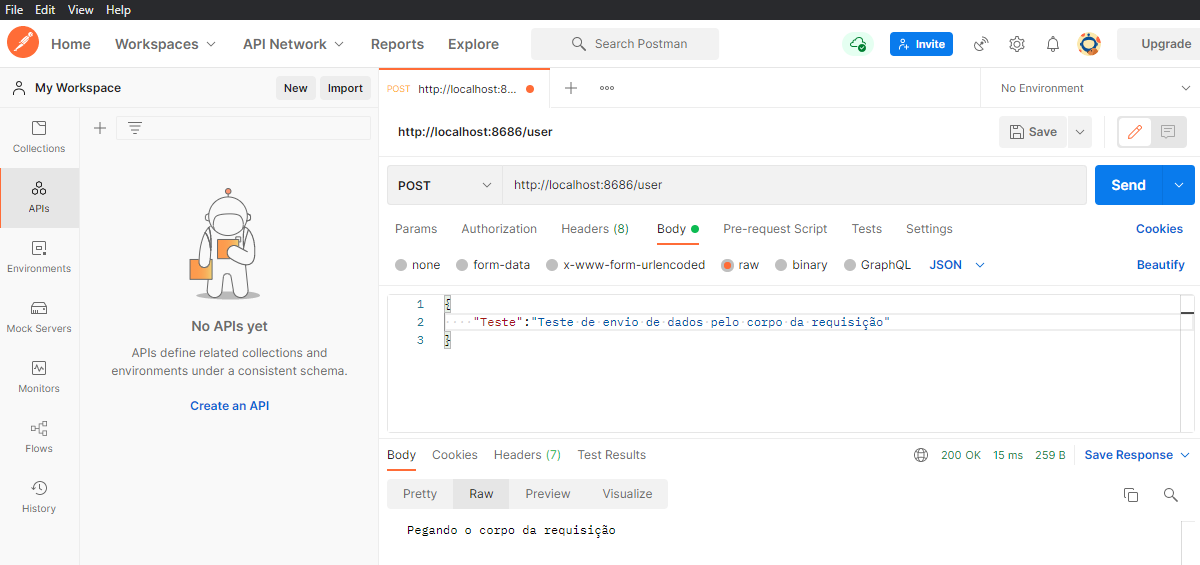
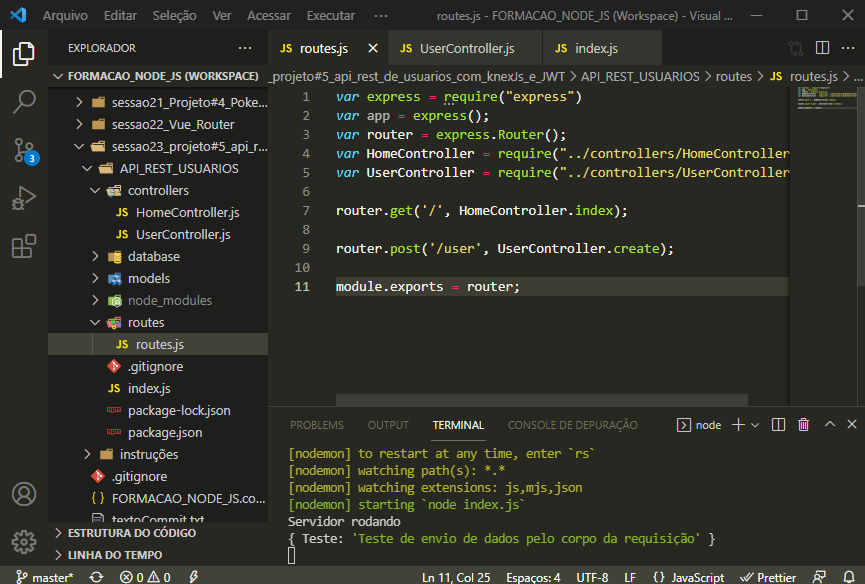
**Instalando o MySql**

1. Navegue até á a página mysql.com entre em Downloads, clique em “MySQL Community (GPL) Downloads”
2. Clique em “MySQL Community Server”
3. Selecione o sistema operacional e clique em “Go to Download Page”
4. Baixe a versão completa.
5. Após baixar clique no arquivo para instalar.
6. Quando abrir o programa, selecione a opção “Server Only” e next
7. Clique em “execute” para finalizar a instalação
8. Clique em next para iniciar a configuração.
9. Clique em next novamente
10. Primeiramente selecine “Develelopment Compute” em Config type.
11. Clieque em next
12. Cadastre um senha e next
13. Clique em next.
14. Clique em execute
15. Clique em finish
16. Clique em next
17. E Finish
18. Vamos linkar o executável do mysql no cmd
19. Vá até a pasta mysql em arquivos de programas
20. Entre na pasta bin
21. Copie o endereço desta pasta
22. Vá em painel de controle, sistema e segurança, sistema, propriedades do sistema, variáveis de ambiente,
23. Em variáveis de usuário para ...
24. Clicke em Path, editar, novo, colar o link que foi copiado, clique em ok , ok e ok.

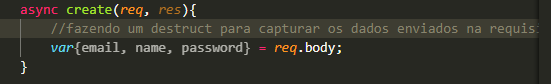
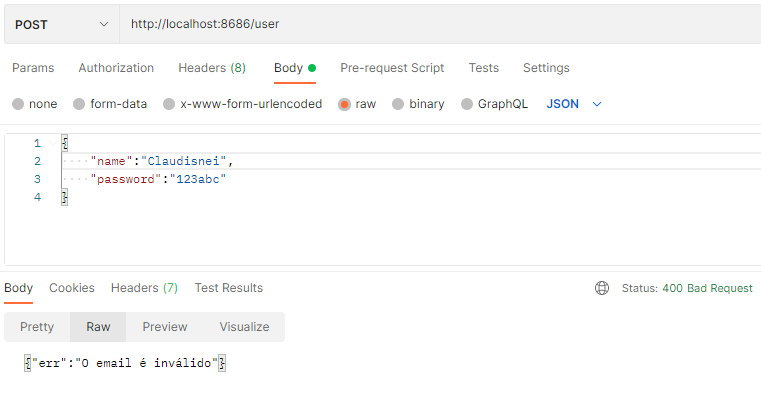
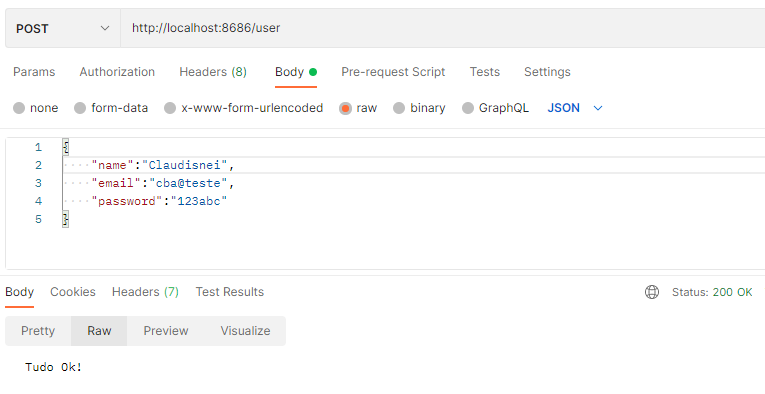
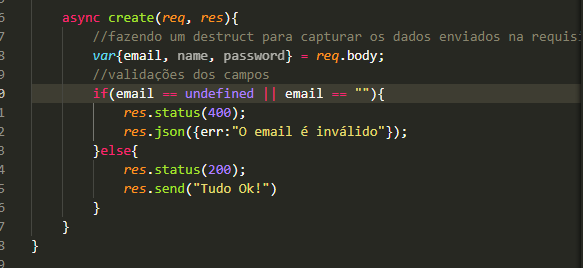
**Começando a crianção com o banco de dados**

1. Sempre que vc for criar uma aplicação com query builder como o Knex, temos que começar com a definição do banco de dados. Vamos utilizar o SGBD HeidiSql.
2. Antes vamos abrir o arquivo de configuração da conexão do banco de dados
3. 
4. Substituía os valores no arquivo connection, pelos dados de sua conexão com o mysql2
5. 
6. Após isso vá no SGBD, neste caso no HeidiSQL e crie um banco de dados
7. 
8. Agora vamos definir a única tabela da api chamada users
9. Crie os campos :
10. 

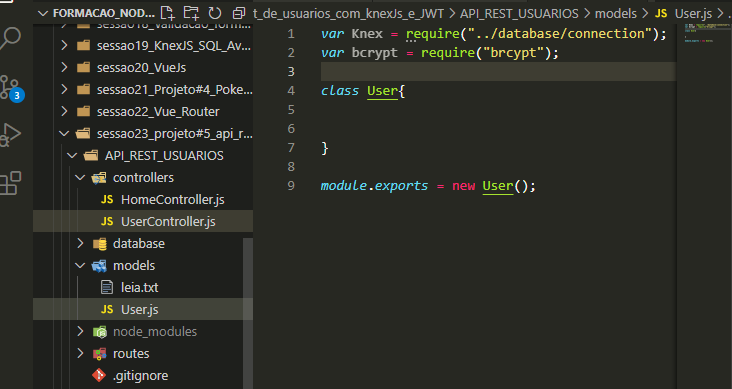
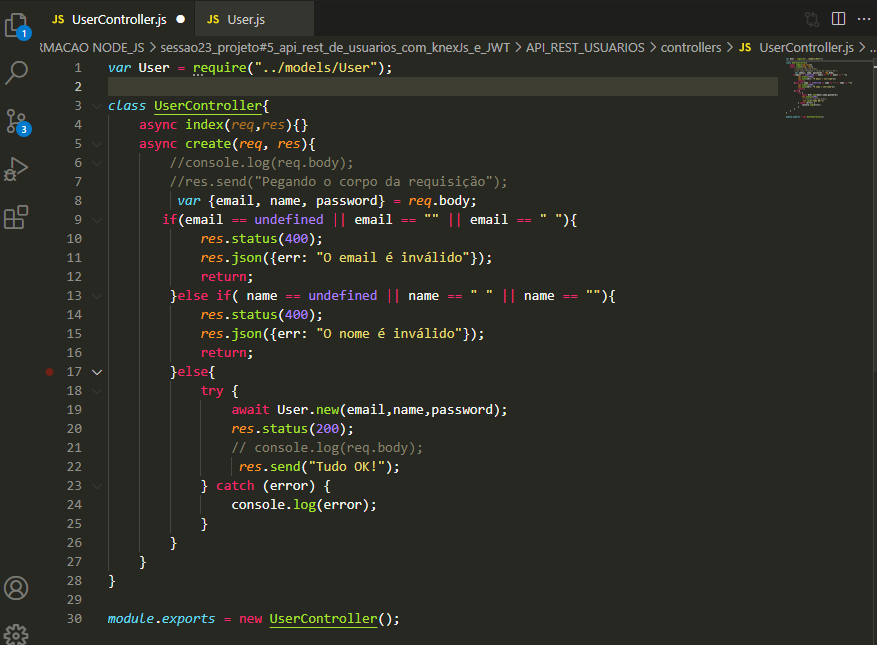
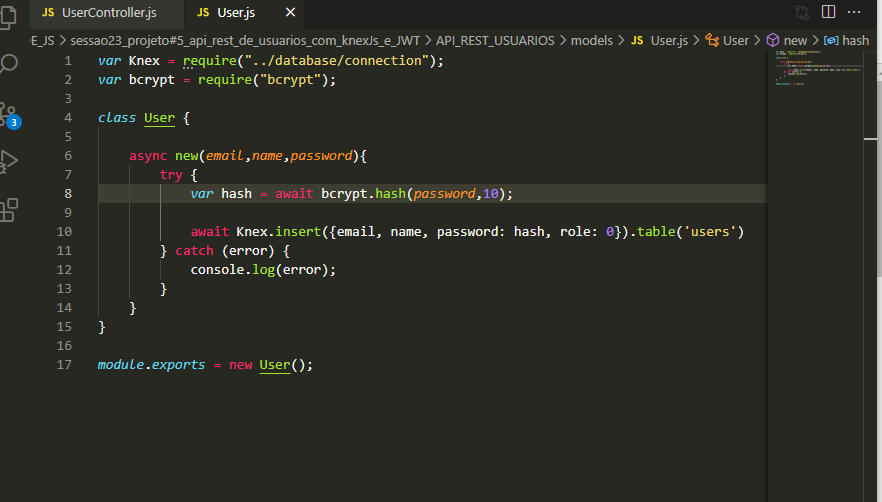
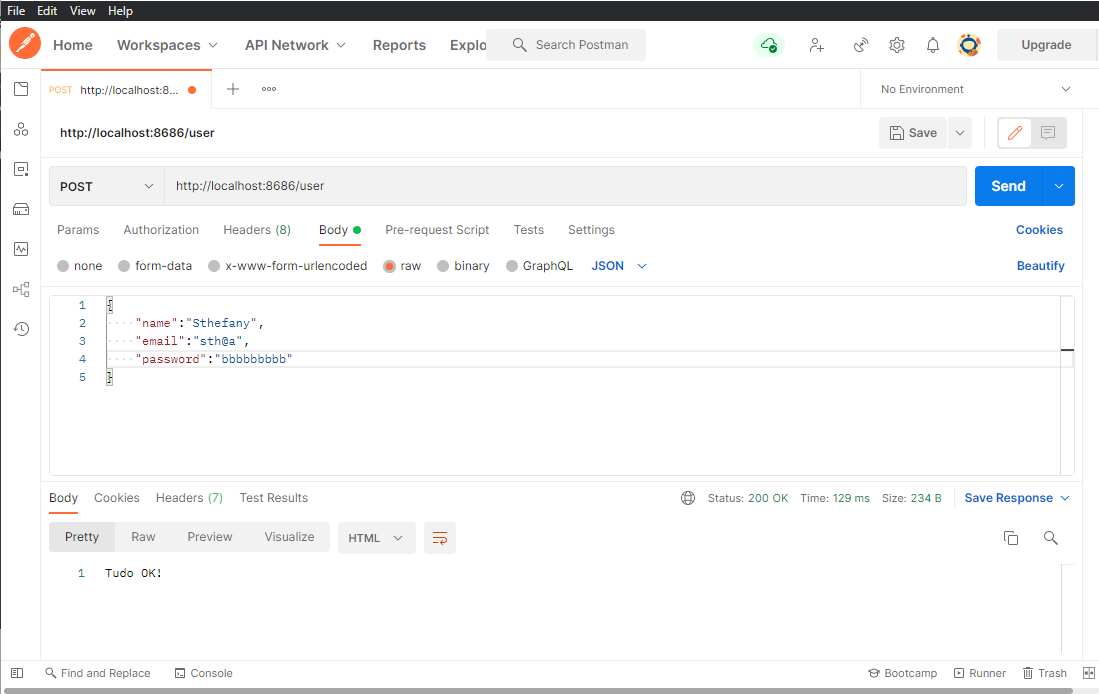
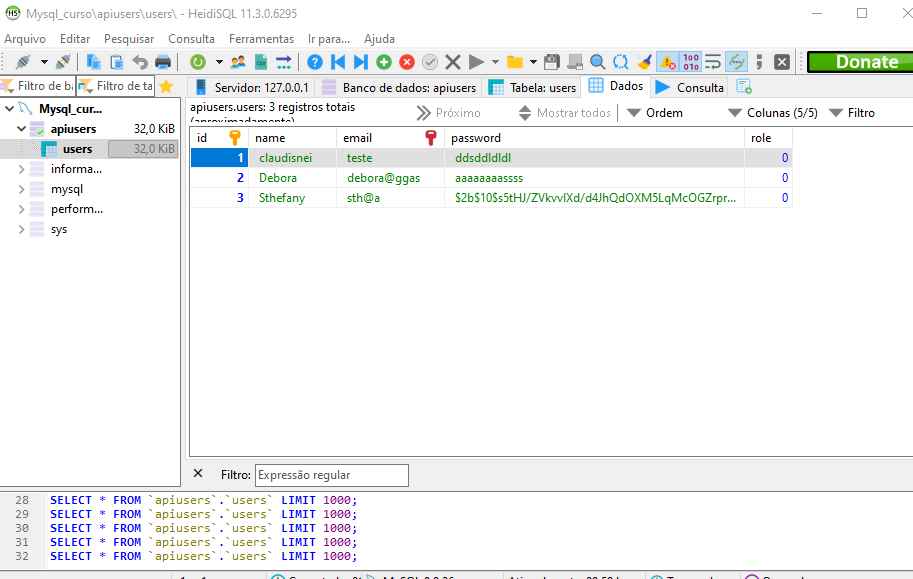
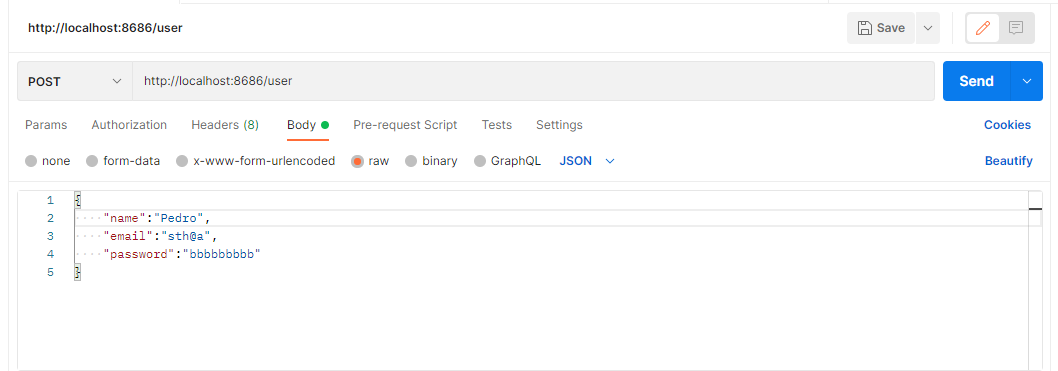
**Se adaptando a estrutura dos Controllers**

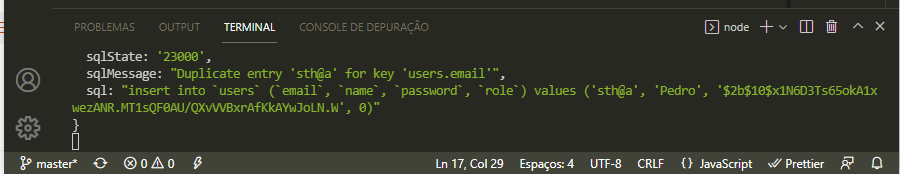
1. na pasta controllers, crie o arquivo UserController.js para criar a lógica de acesso aos dados da aplicação
2. Lembrando que todo end point da aplicação deve ter uma resposta ‘res’
3. No fim do arquivo crie um module.exports para exportar o UserController().
4. 
5. Para usar o controller temos que importa-lo no arquivo de rotas e criar a rota em seguida. Neste caso vamos criar uma rota post já que vamos criar um novo usuário.
6. Router.post(‘/user’), o hast entende que como a rota é post e é passado somente o user, a rota será para salvar um novo usuário.
7. Para testar rotas post, temos que trabalhar om o Postman, por isso vamos instala-lo.
8. Após isso rode novamente o servidor
9. Vá no postman e digite a url <http://localhost:8686/user> na rota post.
10. Aparecerá a mensagem que configuramos no método create no UserController.
11. 
12. Para testar se os dados estão sendo enviados pelo corpo da páigna, no Postman, clique em Body, raw, selecione o formato Json. Escreva uma mensagem em Json. E clique em send.
13. 
14. No console do Vscode aparecerá a mensagem que foi enviada.
15. 

Validação de dados

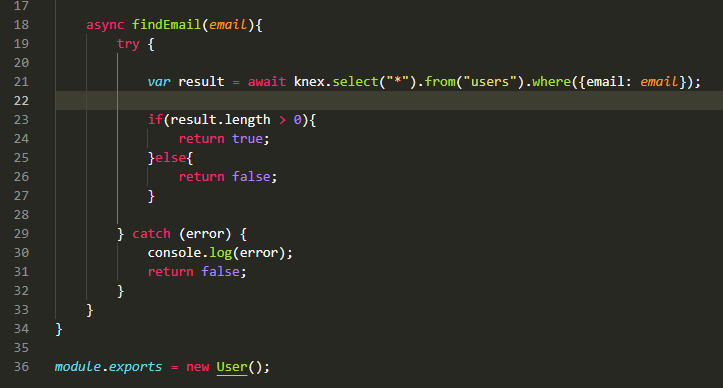
1. Para evitar que o usuário envie dados desnecessários e incorretos para uma rota, os mesmos serão validados.
2. Já foi falado de validação de formulários no curso, por isso vamos fazer uma validação simples.
3. No método create do UserController vamos fazer um destruct, criando três variáveis que receberão os dados enviados pelo corpo da requisição.
4. 
5. Na sequência verificamos se o email é undefined, neste caso será enviado o status code 400 - bad request.
6. 
7. E caso contrário será enviado a resposta de status code 200 - ok
8. 
9. O código ficará assim:
10. 

**Trabalhando com banco de dados**

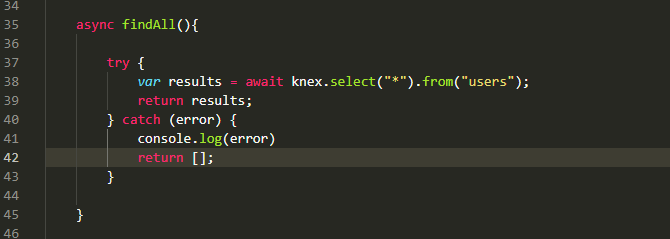
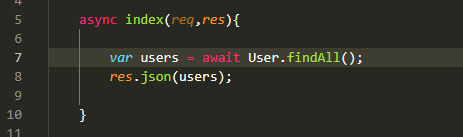
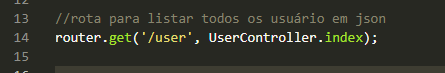
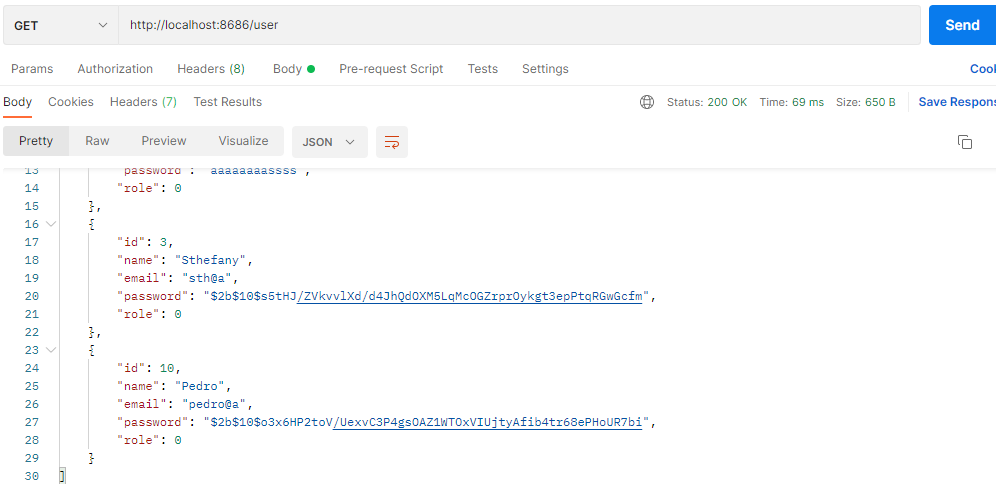
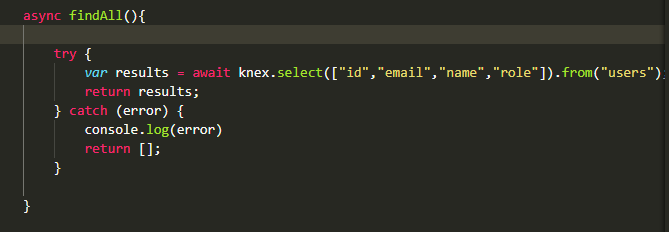
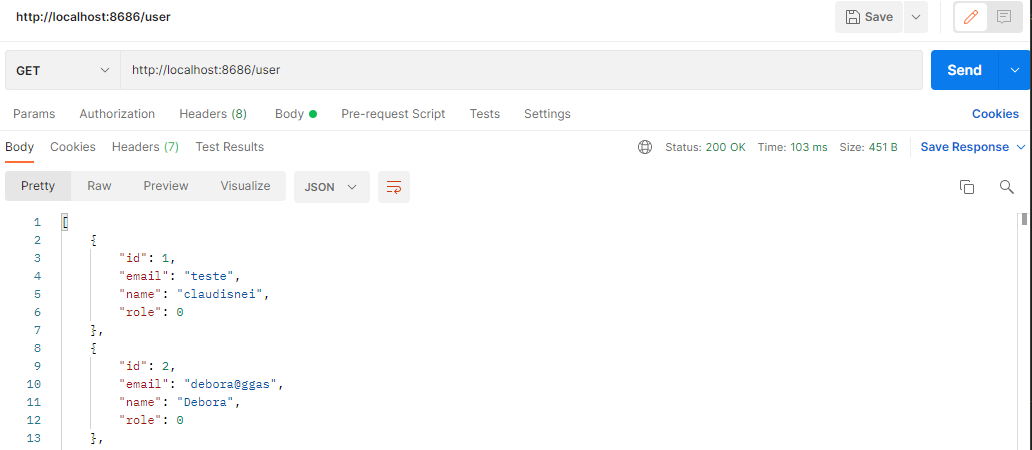
1. Iremos receber dados via requisição para cadastrar um usuário no banco, porém precisamos criptografar a senha.
2. Não podemos salvar a senha do usuário de forma plana, a maneira mais segura é utilizando um hash temos então que transformar a senha em um hash.
3. Para trabalhar com hash no nodejs, vamos instalar a biblioteca bcrypt.
4. Npm install bcrypt - -save.
5. Vamos trabalhar com models utilizando o knex para acessar banco de dados.
6. Vamos criar um model chamado User.js que será uma classe que terá os métodos dela
7. Na classe User vamos carregar o Knex e o bcrypt. O knex irá se conectar ao banco de dados utilizando o arquivo connection da pasta database e bcrypt irá gerar um hash da senha do usuário.
8. 
9. Esse model terá uma função assíncrona chamada new() que receberá um email, password e name para cadastrar, chamaremos o knex.insert passando email, password, name e role como parâmetros.
10. Role por padrão será zero ou seja todo usuário novo será do tipo comum.
11. Será passado ainda como parâmetro dentro de table() a tabela onde os dados serão gravados, neste caso users.
12. Vamos criar um try e cath e colocar await antes de knex para tornar a função assíncrona.
13. 
14. Vamos importar o model User dentro do controler UserController.
15. Após a validação utilizaremos o model User com o método new e os parâmetros email, name e password. Temos que utilizar await pois é uma função assíncrona. Por ser um método dentro de um model, temos que utiliza try para capturar possíveis erros.
16. 
17. O usuário está sendo salvo no banco de dados corretamente, vamos agora utilizar o bcrypt para esconder a senha do usuário em um has.
18. A documentação do bcrypt na internet ensina como utiliza-lo com promisses.
19. Vamos importar o bcrypt no model User.
20. O bcrypt irá chamar a função hash que receberá como parâmetros a senha do usuário e um número que será a quantidade de vezes que a senha será “hasheada”.
21. Vamos criar uma variável hash que irá receber o hash criado pelo bcrypt. O password passado no Knex receberá o hash.
22. 
23. Para testar vamos mandar uma requisição no Postman com os dados de um usuário.
24. 
25. O resultado na tabela users será:
26. 
27. Se tentarmos salvar um novo usuário com um email já cadastrado
28. 

1. irá aparecer o seguinte erro no console, pois foi configurado no banco de dados que o email seria uma chave do tipo Unique, ou seja não pode haver no banco de dados dois email iguais.
2. 

**Método findEmail**

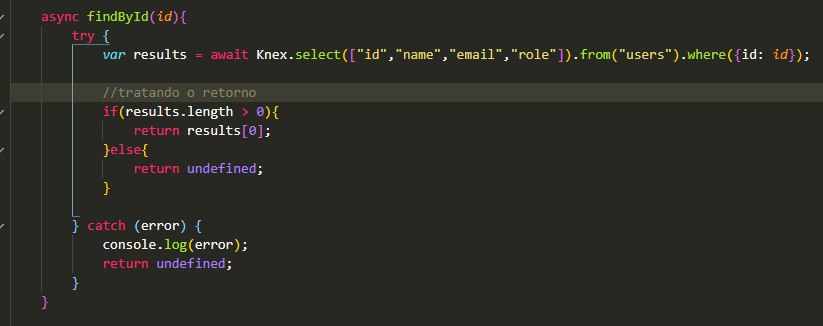
1. Vamos criar um novo método no controller User, chamado findEmail para, antes de salvar no banco de dados, fazer uma busca no BD e verificar se o email já está cadastrado.
2. Vamos criar uma variável result que irá receber o resultado da busca realizada pelo Knex, sendo passado para ele uma string sql. Caso haja um email idêntico já cadastrado o result.length será maior que zero então teremos o retorno como true, caso contrário termos o retorno como false.
3. 
4. No controller UserController vamos criar uma variável emailExists que receberá o resultado do método findEmail.
5. Se o email existir iremos retornar um res.status(406) – Não aceitável e encerraremos a requisição.
6. Caso contrário iremos cadastrar o usuário.
7. 

**Operaçao de listagem de usuários cadastrados**

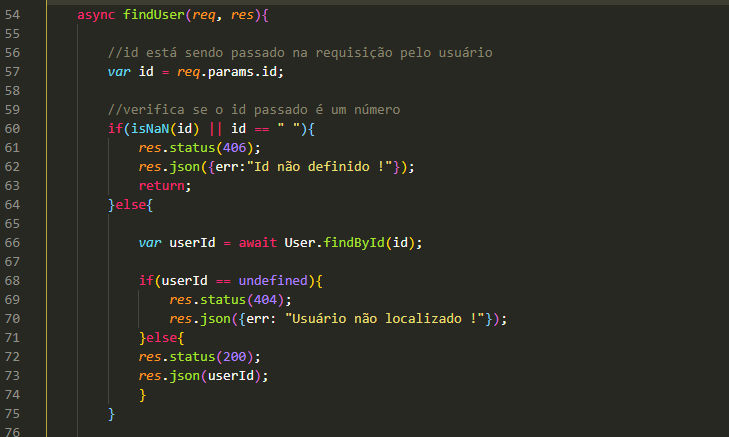
1. Só será permitida se o usuário for um administrador
2. No model de usuário vamos desenvolver um método chamado findAll para listagem de usuários.
3. Neste método vamos utilizar o Knex para localizar todos os registros da tabela users, utilizando sql. O método irá retornar um array com a listagem de usuários ou um array vazio caso ocorra algum erro.
4. 
5. Vamos agora no UserController e vamos utilizar o index para carregar todos os usuários.
6. No método index nos vamos criar um variável users que irá receber a listagem de usuário através do model User e o método findAll.
7. Após vamos criar uma resposta passando a variável users em formato json.
8. 
9. Agora que criamos o método para localizar todos os usuários do banco, vamos criar uma rota para podermos acessar essa informação.
10. No arquivo de routes vamos criar uma rota get, passando ‘/user’ e o método index do UserController, isso irá devolver todos os usários cadastrados.
11. 
12. Ao realizarmos essa consulta vemos que está sendo retornado todos os campos da tabela
13. 
14. porém o campo password não deve ser consultado, por isso vamos voltar ao model User e vamos alterar a busca realizada pelo Knex para ele trazer todos os campos com exceção do campo password.
15. 
16. Com essa alteração o campo password não aparecerá mais na consulta
17. 

**Busca de usuário pelo Id**

**Criando método findById no model User**

1. Vamos agora criar outro método no model User para consultar um usuário pelo ‘id’.
2. No model User vamos criar o método findById() onde será passado para ele um id e o knex realizará a consulta ao banco de dados utilizando o id passado como parâmetro ao método.
3. 

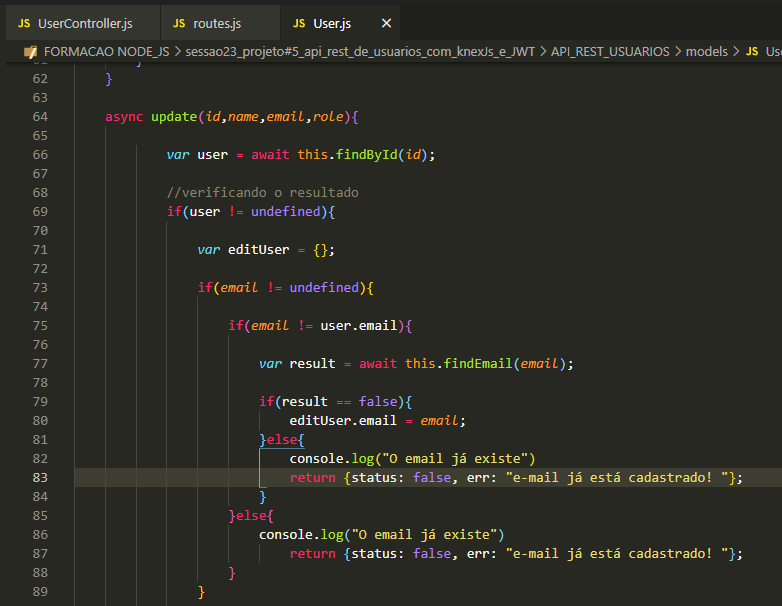
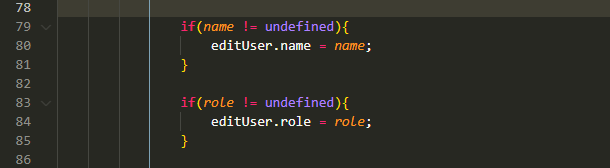
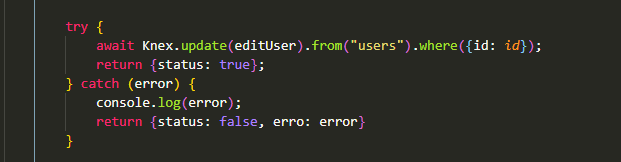
**Criando método findUser no UserController**

1. Após criado o método no model User, vamos criar um outro método dentro do UserController para podermos passar para o usuário a busca realizada no model.
2. Vamos criar o método findUser no UserController que irá receber um id passado pelo usuário no corpo da requisição. Vamos criar uma variável id que irá receber a id que foi passada como parâmetro.
3. Iremos verificar se o id passado é um número, vamos criar uma variável userId que irá receber o resultado da busca realizada pelo método findById do model User.
4. Caso não seja encontrado o usuário, será retornado uma mensagem de usuário não localizado. Se for encontrado será retornado o userId.
5. 

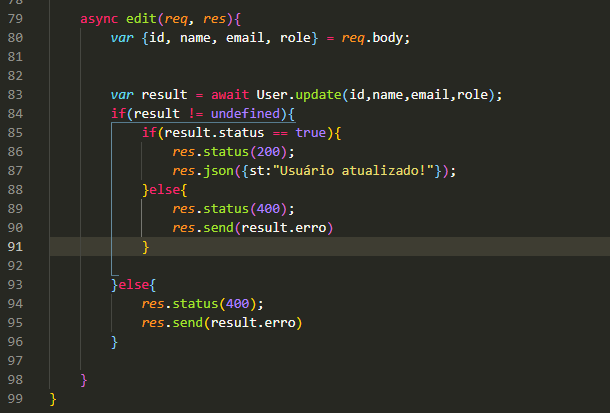
**Edição do Usuário**

1. Vamos criar agora a lógica para editar um usuário cadastrado no sistema.
2. Porém a edição de senha só será possível através de um método de recuperação de senha, portanto a edição só irá atualizar o email, name e hole. Sendo que o email deverá ser verificado para ver ser já está cadastrado.

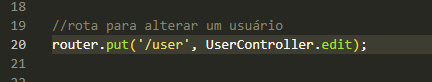
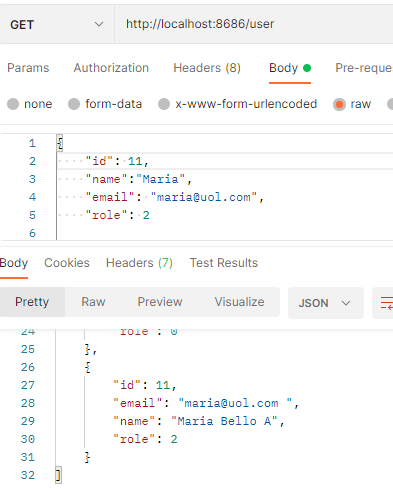
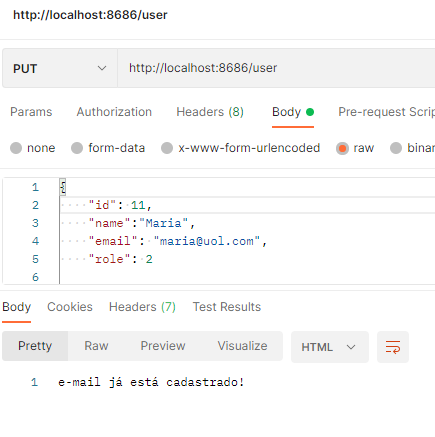
**Criando método update no model User**

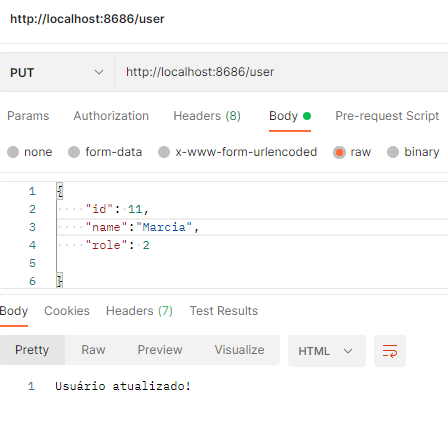
1. O método irá receber como parâmetro os dados que serão atualizados no sistema, porém temos que primeiramente verificar se o usuário existe.
2. Para isso vamos pegar o id passado e verificar através do método findById() se o usário existe.
3. Se o usuário não existir será retornado um json com o status e o erro.
4. Caso contrário iremos preencher um json chamado editUser com os dados passados pelo usuário.
5. Vamos verificar primeiramente se o email passado é diferente do email do usuário atual, caso seja diferente vamos verificar se este email não está cadastrado.
6. Faremos isso pelo método findEmail() se o resultado for false, criaremos um campo email na variável editUser e passamos o email para este campo. Se for true iremos retornar para o usuário um json informando que o email já está cadastrado.
7. 
8. Vamos verificar se o name é diferente de undefined se for vamos criar este campo no ediUser e passar o name para ele.
9. Vamos fazer o mesmo para o campo role.
10. 
11. Após isso vamos chamar o knex com o método update, passando como parâmetro o editUser e o sql, se tudo ocorrer bem retornaremos para o controler o status true para o controller saber que a operação foi realizada com sucesso.
12. 

**Criando método edit no UserController**

1. Primeiramente faremos um destruct para pegar as variáveis passadas pela requisição.
2. Depois criamos uma variável chamada result que irá receber o resultado do método update do model User. Se o result não for undefined. Será verificado o resultado do método.
3. Se for true será passado o status 200 e um json com a mensagem de atualizado com sucesso.
4. Se for false será enviado um status de erro e um json com a erro da varialvel result.
5. Se o result for undefined será mostrado um erro.
6. 

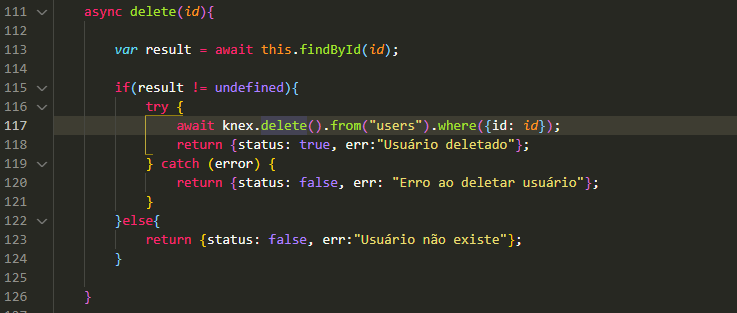
**Criando a rota para realizar o update**

1. No arquivo routes vamos criar uma rota com um put. Passando o UserController com o método edit.
2. 
3. no postman vamos tentar alterar o name do usuário com um email já cadastrado no sistema
4. 
5. Altere o verbo para put e clique em send. Será exibido uma mensagem de e-mail já cadastrado e não fará a alteração no nome.
6. 
7. Porém é possível alterar o nome ou o cargo.

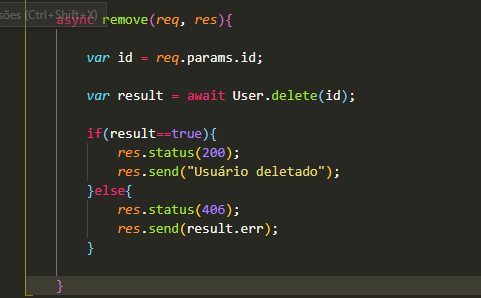


**Deletando um Usuário**

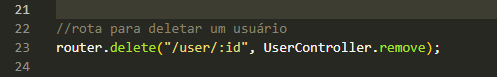
**Método delete no model User**

1. Criamos uma variável result que irá receber a busca do id informado pelo usuário, se não for indefinido chamaremos o método delete do Knex para realizar a deleção do usuário.
2. 

**Criando método remove no UserController**

1. Criamos uma variável id que irá receber a id passada pelo usuário como parâmetro.
2. Criada outra variável result para receber o resultado (true ou false) do método delete do model User.
3. 

**Criando a rota para apagar o usuário**

1. 

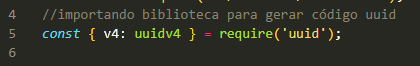
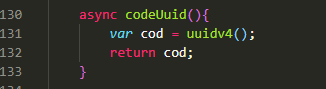
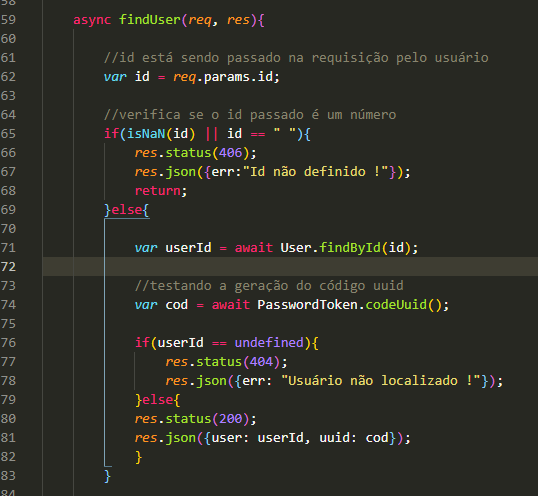
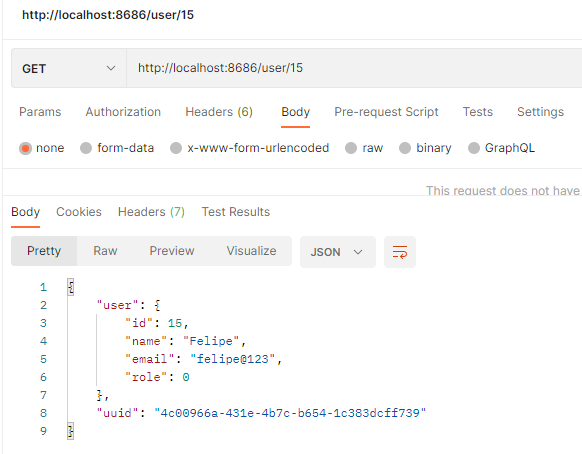
**Recuperando a Senha do Usuário**

1. A lógica de recuperação é a seguinte, quando o usuário clica em esqueci minha senha, é gerado um token para o usuário que só poderá ser utilizado uma vez. Então para redefinir a senha é necessário o usuário passar este token.
2. Então no banco de dados teria uma tabela que teria três colunas uma que guardaria os tokens, outra o id do usuário e outra que guardaria se o token foi utilizado ou não.
3. A melhor forma é trabalhar com UUID ou GUID para gerar o token. Podemos pesquisar no google sobre essas bibliotecas para o nodeJs.
4. Quando usuário clicar em recuperar senha ele receberá um email com um link que abrirá um formulário para recuperação da senha.
5. Este formulário irá conter os campos de nova senha e o token oculto.
6. Quando o usuário enviar a nova senha, será realizado uma consulta no banco de dados para verificar se o token ainda não foi utilizado e neste caso será passado para o cadastro do usuário a nova senha.

**Criando nova tabela no banco de dados**

1. No banco de dados vamos criar uma nova tabela chamada passwordTokens que terá os campos Id do tipo Unsigned não nullo de auto-increment e com primary-key.
2. Teremos outro campo chamado token do tipo varchar de tamanho 200 não nulo.
3. Outro campo será chamado userid do tipo int, Unsigned e não nullo.
4. Clique em salvar para salvar a tabela
5. Para criarmos relacionamento entre dois campos de tabelas, todos os campos devem ser iguais.
6. Vamos clicar sobreo campo userId da tabela passwordtokens, depois clicar sobre chaves-estrangeiras e adicionar
7. Em colunas selecione a coluna que será utilizada no relacionamento
8. Em tabela referente selecione a outra tabela
9. Em colunas referente selecione o campo da outra tabela que fará parte do relacionamento.
10. Em no Update selecione CASCADE para que toda alteração do usuário reflita nas duas tabelas.
11. Em no Delete selecione CASCADE para que quando o usuário for deletado as informações dele na outra tabela tamém sejam apagadas.
12. Clique em salvar.
13. Agora vamos adicionar outro campo na tabela passwordtokens chamado used para informar se o token já foi utilizado ou não. Pois se o token já foi usado uma vez, ele não pode ser usado novamente.
14. O campo será do tipo TINYINT, tamanho 3 e Unsigned.
15. Neste campo será salvo zero ou um. Onde zero informa que o token ainda não foi utilizado e um informa que já foi usado.
16. Clique em salvar para terminar a tabela.

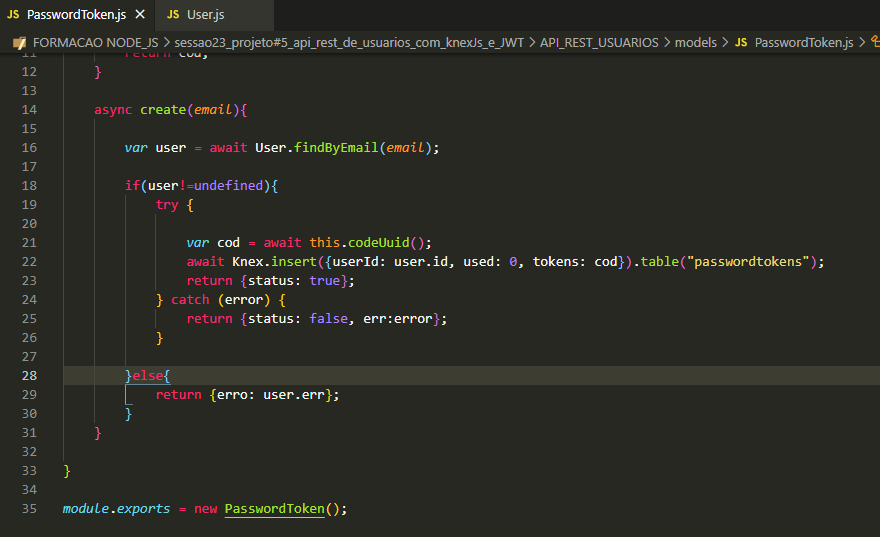
**Instalando a biblioteca uuid na aplicação**

1. Na pasta do projeto digite
2. Npm install uuid.
3. Crie um novo model chamado PasswordToken.
4. Importe a biblioteca para o model .
5. 
6. Crie um método chamado codeUuid para geração do uuid.
7. 
8. No UserController vamos importar o PasswordToken incluir um código para testa a geração do código, após o teste, o mesmo pode ser apagado.
9. 
10. Basta consultar um usuário no postman para verificar a criação do uuid.
11. 
12. Lembrando que uuid é aleatório e mudará toda vez que o usuário for consultado.

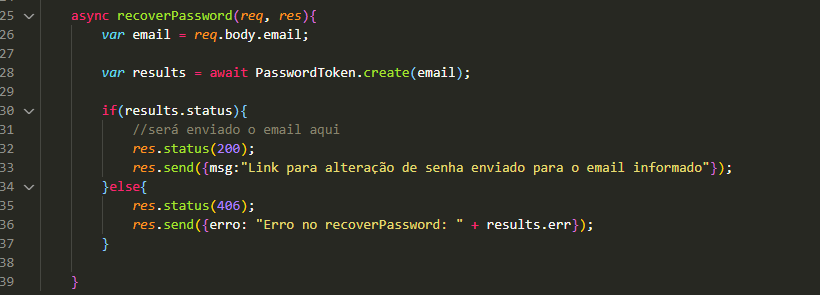
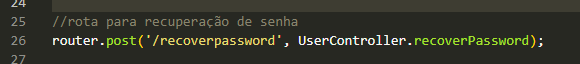
**Geração do Token**

1. Vamos primeiramente importar o Knex direcionando para o arquivo connection.
2. E o model User.
3. 

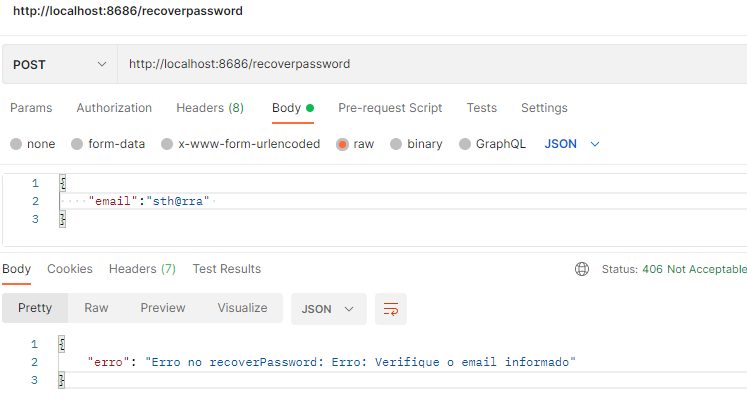
**Criando o método create no model PasswordToken**

1. O método create irá receber o email e irá verificar se o email está cadastrado através do método findByemail do model User.
2. 
3. Se o email está cadastrado no sistema, vamos gerar um token, que será gravado na tabela passwordTokens.
4. Vamos criar a variável token e utilizar o método codeUuid para gerar o token
5. Vamos utilizar o Knex e informar o user\_id, o status de zero para o token pois ainda não foi utilizado e o token
6. 

**Criando um método recoverPassword no UserController**

1. Vamos criar um método recoverypassword que receberá o email que foi enviado pelo usuário pelo corpo da requisição, com este email o usuário quer recuperar a senha.
2. Vamos utilizar o método create do model PasswordTokens que irá receber o email e gerará um token para o usuário no banco de dados
3. 
4. Se for gerado o token, criaremos depois a lógica para envio de email.
5. No arquivo de rotas vamos criar uma nova rota para a recuperação de senha.
6. 

**Teste no postman**

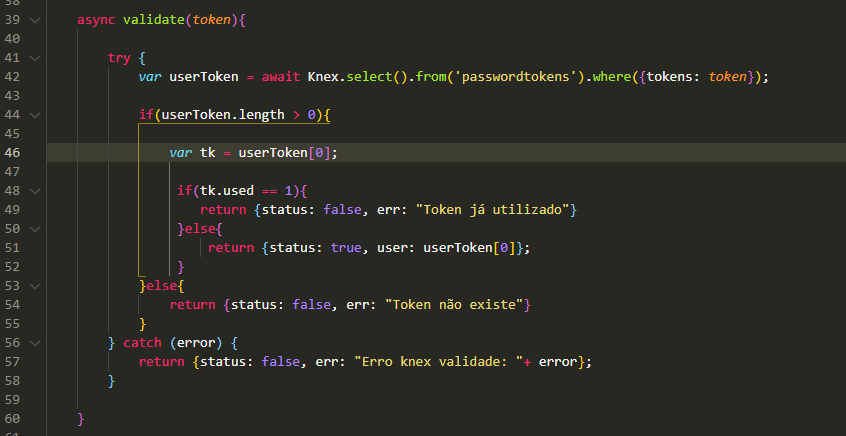
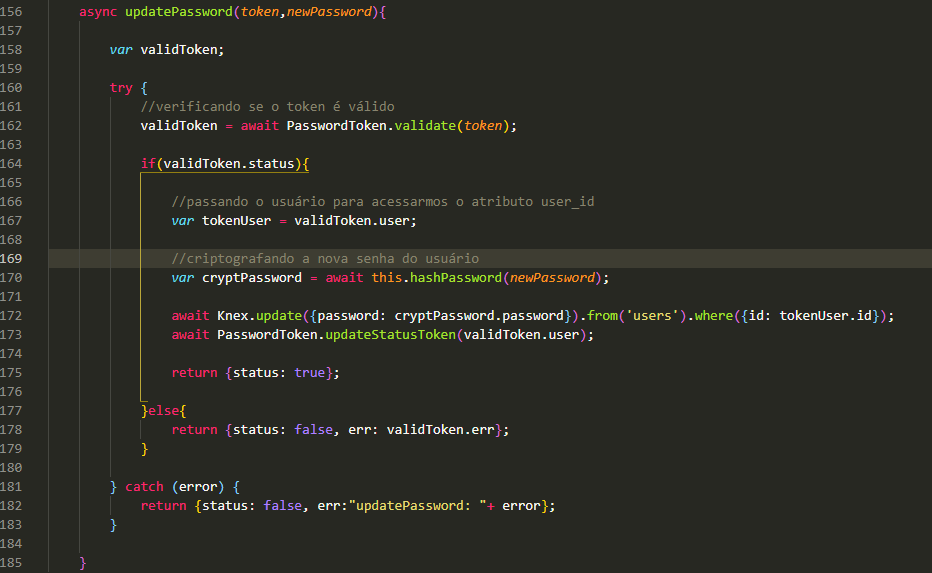
1. Com o servidor rodando vá no postman e digite a rota.
2. <http://localhost:8686/recerverpassord>.
3. Selecione o verbo post, clique em body, clique em raw,
4. Neste campo digite um json com o email do usuário que vc quer que seja realizado a recuperação de senha.
5. Para testar digite um email errado, será retornado uma mensagem de erro.
6. 
7. Agora informe um email válido do banco de dados. Será informado que foi enviado um link para o usuário realizar a troca da senha.
8. 

**Alterando a senha do Usuário**

**Método changePassword no Usercontroller**

1. Neste método vamos fazer um destruct onde iremos capturar as variáveis ‘token’ e ‘newPassword’ passadas pelo método post via requisição.
2. As variáveis serão passadas como parâmetro para o método updatePassword do model User.
3. Se o resultado for diferente de undefined será verificado o status se for true significa que a senha foi alterada e o status do token alterado.
4. 

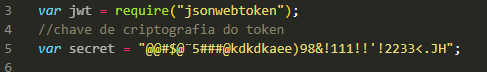
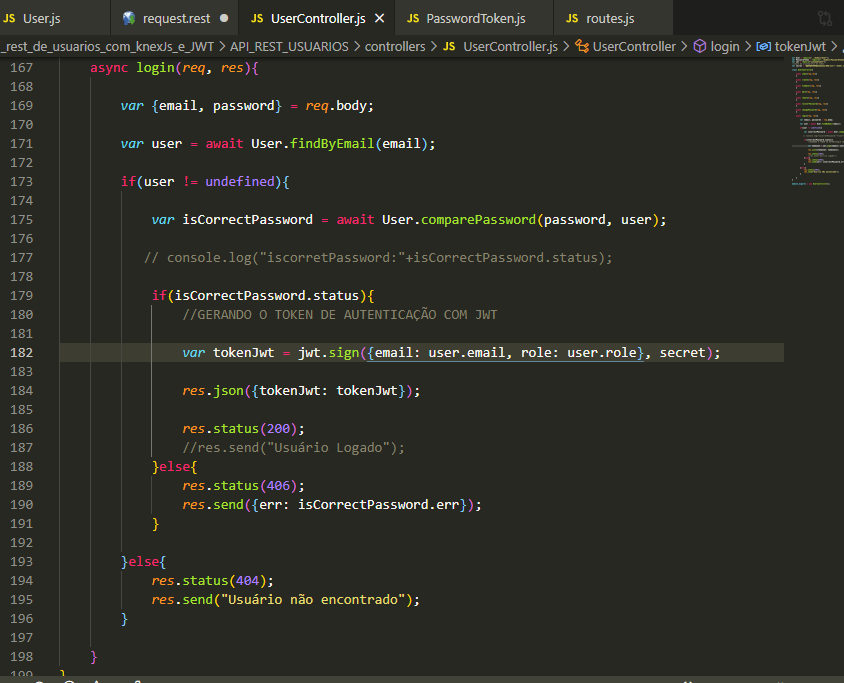
**Método upadtePassword no model User**

1. Este método recebe como parâmetro o token e a nova senha do usuário.
2. Criamos uma variável validToken receber o resultado do método validate no model PasswordToken.
3. O método validate recebe o token que foi passado como parâmetro e através do Knex é realizado um select na tabela passwortokens verificando se existe algum token idêntico ao informado.
4. Se for encontrado um token idêntico ao informado, é feito uma verificação se o token já foi usado, verificando se o atributo used é zero ou um.
5. Caso seja zero o o registro da tabela será retornado, caso contrário será retornado a mensagem de token já utilizado ou “token não existe”.
6. 
7. Na sequência no método updatePassword,
8. É feito uma verificação da variável validToken, para ver se o status retornado pelo método validate.
9. Se o status for true, criamos uma variável tokenUser que irá receber o registro userToken passado no retorno do método validate.
10. Criamos um método chamado hashPassword que recebe a nova senha e retorna um hash.
11. Então passamos para hasPassword a senha passada pelo usuário e recebemos em cryptPassword o hash.
12. Utilizamos agora o Knex.update para atualizar a tabela users passando para o usuário atual a nova senha.
13. Utilizamos também o método updateStatusToken do módel PasswordToken, que recebe o userToken passado como retorno do método validToken.
14. 

**Autenticação JWT e login**

1. Vamos instalar a bibliote jwt para autenticação digite:
2. Npm install –save jsonwebtoken

**Método login no UserController**

1. Vamos agora importar a biblioteca jwt no UserController
2. Vamos também criar uma chave de criptografia do tokenJwt.
3. 
4. Este método irá receber o email e a senha que serão passado pela requisição.
5. Criamos uma variável user receberá o retorno do método findByEmail do model User, verificando se o email informado está cadastrado no sistema.
6. Se a variável user for diferente de undefined significa que achamos um usuário que possui o email passado, agora vamos comparar para ver se a senha que foi passada pelo usuário é igual a que está cadastrada no banco de dados.
7. Para isso vamos criar uma variável chamada isCorrectPassword e utilizar o método comparePassword do model User, que recebe a senha passada por requisição e o user que foi encontrado no método findByEmail.
8. Se isCorrectPassword for true significa que as senhas são iguais.
9. Criamos então a variável tokenJwt que receberá um token de autenticação. Para isso utilizamos o jwt.sign que irá receber o emai, o cargo do usuário e a chave secret criada anteriormente.
10. E enviamos o token por resposta.
11. 

**Método comparePassword do model User**

1. Este método recebe como parâmetros a senha passada pelo o usuário e o usuário encontrado no método findByemail dentro do método login, visto anteriormente.
2. Criamos então uma variável chamada truePassword que irá receber o resultado do método compare do bycrypt, este método recebe como parâmetro a senha passada pelo usuário e o password da variável user, que é a mesma do banco de dados.
3. 

**Rota para login**

1. 